



Persistent

Core Java: Class Packaging

Persistent Interactive | Persistent University



Objectives :

- At the end of this module, you will be able to understand :
 - Use of package
 - Types of packages
 - Creating packages
 - Using packages in applications
 - Static import

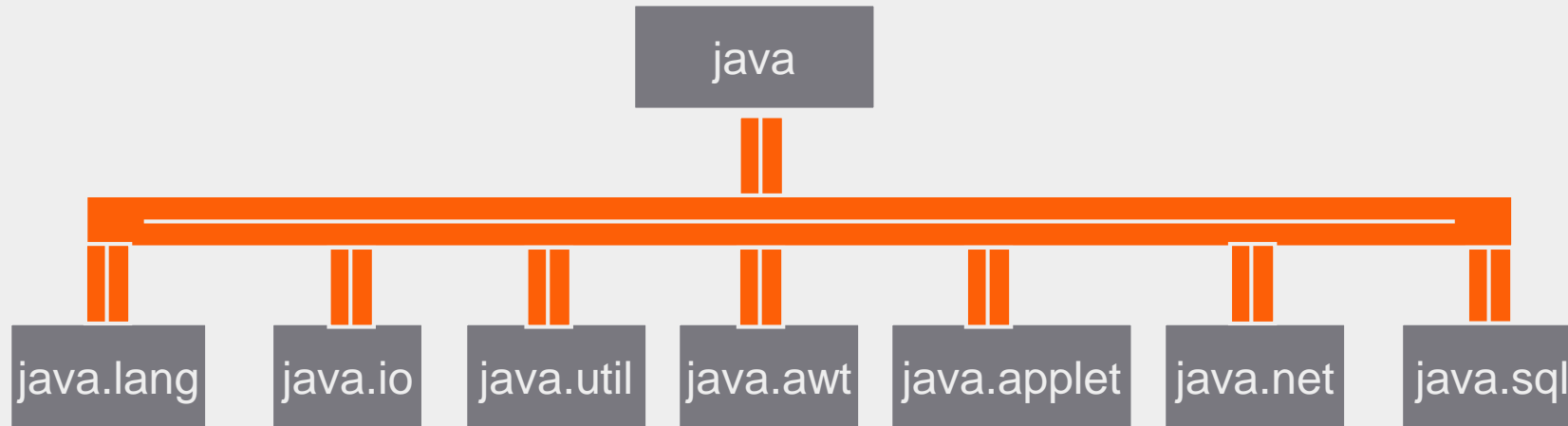
Understanding usage of packages

- "A package is a namespace that organizes the a set of related classes and interfaces. It also defines the scope of a class. An application consists of thousands of classes and interfaces, and therefore, it is very important to organize them logically into packages.
- By definition, package is a grouping of related types providing access protection and name space management."
- Packages enable you to organize the class files provided by Java.



Packages in Java

- The packages in Java can be nested. The 'java' is the root package for all the packages of Java API. Below is the partial hierarchy of Java API.



The classes of a package are qualified with package name

Packaging...

- The first question in building an application is "How do I divide it up into packages?".
- For typical business applications, there seems to be two ways of answering this question.
- Package-by-feature uses packages to reflect the feature set. It places all items related to a single feature (and only that feature) into a single directory/package.
- Package-by-layer implementation is spread out over multiple directories.

Packaging-by-feature

- This results in packages with high cohesion and high modularity, with minimal coupling between packages. Items that work closely together are placed next to each other.
- In package-by-feature, the package names correspond to important, high-level aspects of the problem domain. For example, a drug prescription application might have these packages :
 - com.app.doctor
 - com.app.drug
 - com.app.patient
 - com.app.prescription
 - com.app.report
 - com.app.security
 - com.app.webmaster
 - com.app.util
 - and so on...

Packaging-by-feature

- Within each package are all (or most) items related to that particular feature (and only that feature).
- For example, the com.app.doctor package might contain these items :

DoctorAction.java - an action or controller object

Doctor.java - a Model Object

DoctorDAO.java - Data Access Object

database items

user interface items (perhaps a JSP, in the case of a web app)

Packaging-by-layer

- The competing package-by-layer style is different. In package-by-layer, the highest level packages reflect the various application "layers", instead of features, as in :
 - com.app.action
 - com.app.model
 - com.app.dao
 - com.app.util

Significance of import keyword

- The classes in the packages are to be included into the applications as required.
- The classes of a package are included using import keyword
- The import keyword followed by fully qualified class name is to be placed before the application class definition.

```
import packagename.classname ;  
    // Class Definition.
```

```
import java.io.*;  
import java.util.Date;  
    // Class Definition
```

- Above indicates to import all the classes and a specific class from a package.
- All the classes in java.lang package are included implicitly.

User defined packages

- Creating a user-defined package

```
package <package_name>
// Class definition
public class <classname1>
{
    // Body of the class.
}
```

```
package vehicle;
public class Car
{
    String brand;
    String color;
    int wheels;
}
```

User defined packages (continued)

- You can include a user-defined package or any Java API using the import statement.
- The following syntax shows how to implement the user-defined package, vehicle from in a class known as MarutiCar






```
import vehicle.Car;  
  
public class MarutiCar extends Car  
    // Body of the class.
```

Access modifiers

An attribute that determines whether or not a class member is accessible in an expression or declaration.

Type of Access Modifier	
private	Members are accessible only to the class.
package / friendly	Members are accessible only to the class and other classes within that package. This is the default access specifier.
protected	Members are accessible only to the class and its subclass(es).
public	Members are accessible to the class and to other classes.

Access modifiers

Access modifiers	Class	Package	Subclass within the package	Subclass outside the package	World
private					
default/No Modifier					
protected					
public					

Choosing a suitable Access Control Modifier for Classes and Fields

- As a guideline when choosing an access level it's generally best to use the most restrictive level possible for classes, fields and methods.
- If class having the default access control modifier then never make any member public or protected owing to the fact the class itself can't be used outside the package.
- For fields, the thumb rule is –
 - make all instance fields 'private' and have getters/setters to access/modify (but not for all) them.
 - make most of your static fields as 'public' as they are class variables and meant to be accessed from outside either on the class name (preferred way) or on any instance.

Choosing a suitable Access Control Modifier for Methods

- Make a method public only if you can't make public/protected/default
 - private: no commitment towards others.
 - can be changed anytime as long as it doesn't hurt the functioning of any public/protected/default member.
 - default:
 - Commitment towards all the classes in the same package as they can have their implementation dependent on those members.
 - protected:
 - use when you want your sub-classes to use it.
 - public: **be very careful**
 - as clients of the class can use them and hence once you make something public you have to support that for the clients of your class till eternity, can't make it deprecated easily if required.

Static import

- Introduced in JDK 1.5.
- To use static data members of a class in another class requires the syntax as `classname.variablename` or `classname.method()`.
- Static import allows the java developer to access static data members & member functions of a class without using `classname`.
- Less coding will be required if frequent usage of static members is required.

Syntax for static import

- System class has out variable declared as static.
- To import this variable by using static import following syntax will be used.
- After static import , this can be used as

```
import static java.lang.System.out;  
out.println(//print on console);
```

Use of static import

```
import static java.lang.System.out;
import static java.lang.Integer.parseInt;

public class StaticImportDemo1 {
    public static void main(String[] args) {
        out.println("Hello World");

        String str = "10";
        int num1 = parseInt(str);

        out.println(num1);
    }
}
```

Summary :

With this we have come to an end of our session, where we discussed about

- Packages in java

At the end of this session, we see that you are now able to

- Use packages
- Create your own packages
- Specify access modifier
- Use static import

Appendix



References

Thank you

Reference Material : Websites & Blogs

- http://www.tutorialspoint.com/java/java_packages.htm
- <http://www.javatpoint.com/package>
- <https://docs.oracle.com/javase/tutorial/java/package/packages.html>

Reference Material : Books

- ***Head First Java***
 - *By: Kathy Sierra, Bert Bates*
 - *Publisher: O'Reilly Media, Inc.*
- ***Java Complete Reference***
 - *By Herbert Schildt*



Thank you!

Persistent Interactive | Persistent University

